

Islington College



Information System

CC4002NI

COURSEWORK

Submitted By:

Manish Giri(16034959)

Date: JULY 18, 2017

Submitted To:

SukritShakya /YunishaBajracharya

Word Count: 4532

Semester: Summer

PROPOSAL:

PURPOSE:

The ultimate goal of the course work is to present a program of “byte adder” on the basis of “bit adder”. Through python programming language I can present this program in this report. Python is a powerful high-level, object-oriented programming language created by Guido Van Rossum .Not only python-code but its algorithm, pseudo code and data structure will also be equally explained and demonstrated in this coursework for the byte adder program.

Firstly, starting with the algorithm (i.e step by step method to write a program in a human friendly English language) followed by introduction of pseudo code which will help any programmer to understand more about the program that's written in python or any other language. In conclusion, reasonable information structures like primitive or complex data structure will likewise be chosen and testing for the program will likewise be finished with its testing heading and individual screenshots of the testing.

PROBLEMS STATEMENT:

Python is a different programming language so it's an issue to compose coding in python as it has its own sentence structure, capacities numerous more guidelines to be taken after. This issue will be comprehended by the assistance of our module leader and tremendous measure of research as well as the assignment that we had previously done which was given by our teacher will also come in aid.

AIMS AND OBJECTIVES:

Aims and goals of this coursework are as follows:

- ❖ To choose appropriate information structures to present the data in python language.
- ❖ To show the program with reasonable testing and mistake rectification.
- ❖ To create a well documented report for the coursework.
- ❖ To construct a byte adder based on the bit adder principle.
- ❖ To create an descriptive image of the logic gate model.
- ❖ To specify an algorithm for integer addition based on binary operation for byte adder.

PROPOSED APPROACH:

The approach section is the heart of most proposals. It is vital because it describes the work to be done and details the methodologies or tasks for performing the work

SCOPE OF THE PROJECT:

At the end of the report, this document will give a full detailed report of a byte adder based on bit adder that will include its coding-part, algorithm-part, pseudo code-part, graphical representation of the logic gates that are used ,logs of errors that came through out the coding part and also its rectification-part. Also the scope of the project includes web applications, desktop applications, hardware programming, data science, machine learning and deep learning.

TARGET AUDIENCE:

The main aim of this project is to target unemployment and uneducated people living there who are spending their life doing nothing. By the help of this project we are sure that an individual can develop their own skills. This project wants to make feel them that an individual is capable of doing something if they provided an opportunity.

SOFTWARES AND HARDWARES REQUIRED:

To build up the program, I require python programming IDLE (Integrated Development and Learning Environment). All my python coding for the program will be done in the environment. Different virtual products or applications such as logic.ly will be required to alter the report and additionally present the logic gate model.

PLANNING AND TIMESCALE:

Starting, I will finish the research work about covering all the prerequisite for both the report and programming portion of this coursework. After that the gathered assets will be investigated and compressed by the necessities of the coursework. My next objective will be to build up the program which will likewise have some procedure and hence take maximum amount of my time. I will compose the calculation and pseudo code at that point compose the program as per both of it. After that I will show the program with appropriate testing and error-correction. At long last, I will compose the report and place every one of my works set up to submit. Prior to that I will counsel and check for inputs from my module pioneer for any conceivable change in my coursework and after that I will present my work to the RTE bureau of the information system module.

Table Of Contents

Contents

PROPOSAL:	2
PURPOSE:	2
PROBLEMS STATEMENT:	2
AIMS AND OBJECTIVES:	2
PROPOSED APPROACH:	3
SCOPE OF THE PROJECT:	3
TARGET AUDIENCE:	3
SOFTWARES AND HARDWARES REQUIRED:	3
PLANNING AND TIMESCALE:	4
Table Of Contents	5
Table Of Figures	6
List of Tables	7
Introduction	8
This coursework was given as an assignment which totally focuses on the python, a programming language	8
Discussion and Analysis	8
Full adder	11
8-Bit Adder	12
The 8-Bit Adder Principle	12
Algorithm	17
Pseudo code:	18
Flowchart:	25
Data Structures	26
Len (len)	26
String (str)	26
Integer (int)	26
Input (input)	26
Bin (bin)	27

List (list).....	27
eval()	27
print()	27
Program.....	27
Main.py:.....	27
FullAdder.py:.....	32
Research.....	35
Books	35
Journals	36
Websites.....	36
Testing	37
Alphabet User input.....	37
Passing Negative Value	37
Out Of Range Value From User.....	38
User Input of 1 and 1	38
Value crosses from stetted limited value and passed with negative integer	39
Both value is 0.....	39
Conclusion	40
Bibliography	41

Table Of Figures

Figure 1 Graphically Represented Model	10
Figure 2 Full Adder Circuit.....	11
Figure 3 Block Diagram and Truth table of Full Adder.....	12
Figure 4 OR gate truth table	13
Figure 5 NOT gate truth table	13
Figure 6 Nand gate truth table.....	14
Figure 7 NOR gate Truth table	15
Figure 8 XOR gate Truth table.....	16
Figure 9 Main.py Code	30

Figure 10 Binary Decimal.py	31
Figure 11 FullAdder.py	32
Figure 12 LogicGates.py	33
Figure 13 Alphabet Test	37
Figure 14 Negative Value Test	37
Figure 15 Out of range Value Test	38
Figure 16 input 1 and 1	38
Figure 17 Value crosses from setted limited value and passed with negative integer	39
Figure 18 both value 0 test	39

List of Tables

Table 1 Alphabet Test	37
Table 2 Negative Value Test.....	37
Table 3 Out of range Value Test.....	38
Table 4 1 on 1 input	38
Table 5 Valuue corss passed with negative integer	39
Table 6 Both value is 0	39

Introduction

This coursework was given as an assignment which totally focuses on the python, a programming language

Python is a computer programming language that lets you work more quickly than other programming languages. It's a high level programming dialect which was first released in 1991 by a person named Guido Van Rossum. Coders can be more active and productive with python as its dynamically typed language

An algorithm is a procedure or formula for solving a problem, based on conducting a sequence of specified actions. A computer program can be viewed as an elaborate algorithm. In mathematics and computer science, an algorithm usually means a small procedure that solves a recurrent problem. Algorithms are widely used throughout all areas of IT.

Pseudo code is a description of coding in a readable language for client or other people to understand the code. With the help of pseudo code people can understand easily what's going on with the program.

Discussion and Analysis

The coursework given was a individual coursework so there were no meetings or discussion with anyone except with teachers for guidance purposes.

The lectures, tutorials and the lab conducted in the college were great aid in the completion of the coursework as the basic knowledge of Python programming language was acquired.

Different types of excercises that were given throughout the classes also came to be great help as they helped to know the functionality of Python language even better.

For python to work we need to download the environment from the website

<https://www.python.org/> which includes the python IDLE mode where the environment is set to start coding. The documentation that comes along with the downloaded product helps to utilize the environment to its full that is with the fact that a person has knowledge of python programming language

<https://logic.ly/> is the web application which provided a great aid to create the model gates. The web application is super user friendly and provides set of tools along with hints and tutorials that help in the graphical representation of circuits and other graphical representations.

The coursework is about a byte adder which is based on bit adder. A bit is a binary digit and is the smallest unit of data in a computer. A bit has a single binary value either 0 or 1. A byte is a unit of digital information that most commonly consists of eight bits.

Model

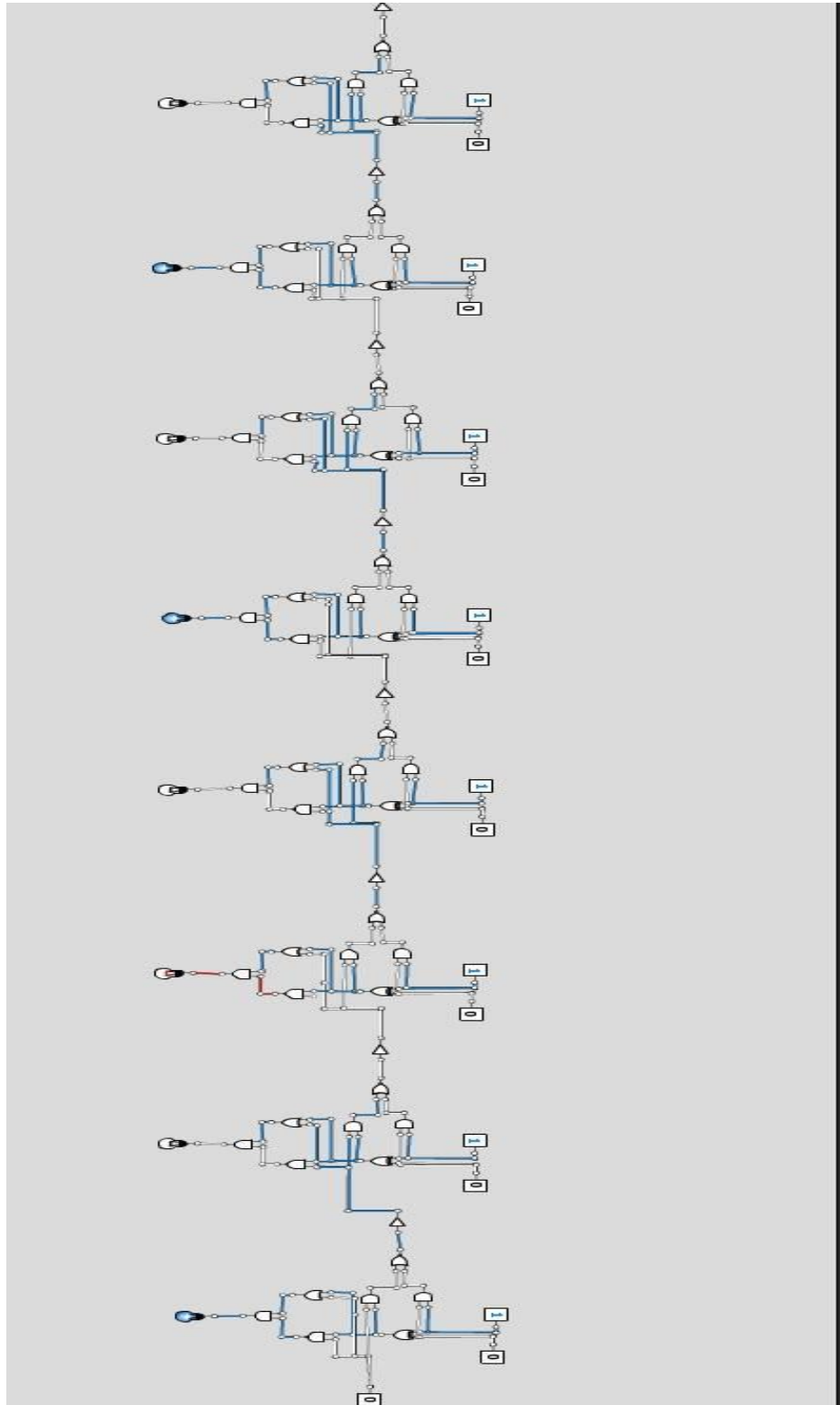


Figure 1 Graphically Represented Model

In the above graph, one half adder and seven full adders are joined to make a full byte adder. To start with, we developed a half adder. It doesn't comprise of carry in then we put the underlying value to 00. Utilizing the complete from the half adder we connected it to the following full adder which has carry in. The value that we input in the outline are of four mixes (00, 11, 10, 01) separately and afterward repeated it for two times. The glowing bulb shows the output with given data sources.

Full adder

This adder is difficult to implement than a half-adder. The difference between a half-adder and a full-adder is that the full-adder has three inputs and two outputs, whereas half adder has only two inputs and two outputs. The first two inputs are A and B and the third input is an input carry as C-IN. When a full-adder logic is designed, you string eight of them together to create a byte-wide adder and cascade the carry bit from one adder to the next.

With the truth-table, the full adder logic can be implemented. You can see that the output S is an XOR between the input A and the half-adder, SUM output with B and C-IN inputs. We take C-OUT will only be true if any of the two inputs out of the three are HIGH.

So, we can implement a full adder circuit with the help of two half adder circuits. At first, half adder will be used to add A and B to produce a partial Sum and a second half adder logic can be used to add C-IN to the Sum produced by the first half adder to get the final S output.

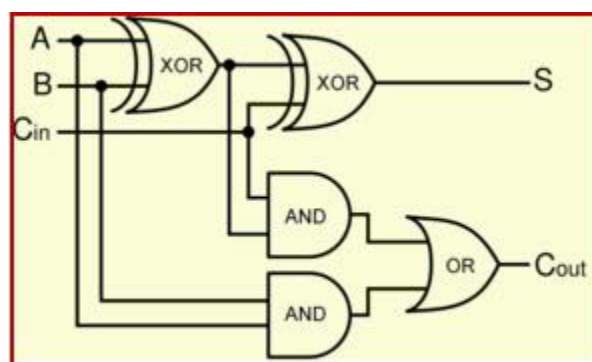


Figure 2 Full Adder Circuit

8-Bit Adder

Binary adders are digital circuits often used in computers for basic arithmetic operations. Using the digital library in the standard Modelica library, we have constructed an 8-bit adder that takes two 8-bit integers and calculates their sum.

The 8-Bit Adder Principle

The 8-bit adder adds the numbers digit by digit, as can be seen in the schematic diagram below. In this example, the integers 170 and 51 represent input a and b, respectively, and the resulting output is the sum 221. The first adder does not have any carry-in, and so it is represented by a half adder (HA) instead of a full adder (FA).

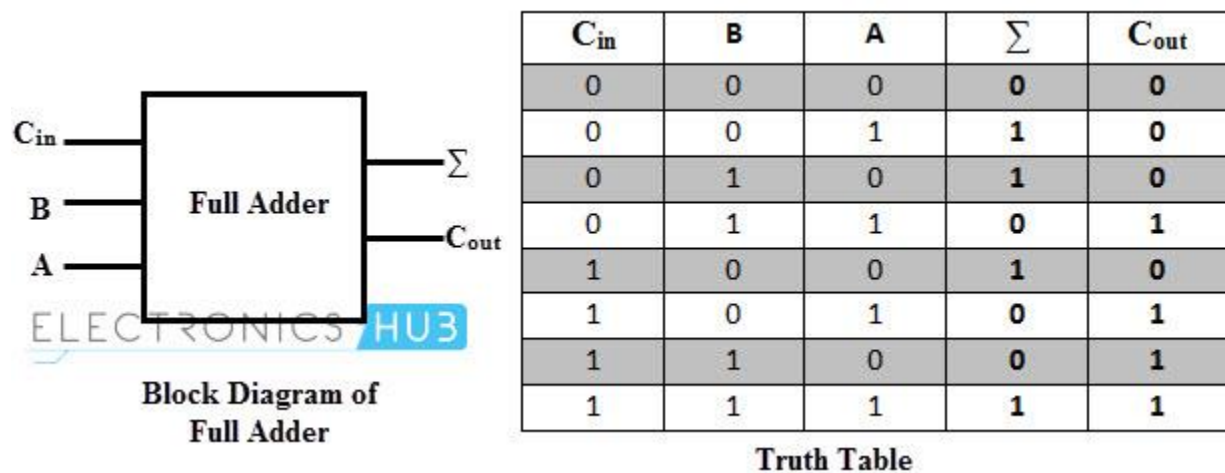


Figure 3 Block Diagram and Truth table of Full Adder

Logic Gates

- OR Gate:

The OR Gate contains two or more than two input values which produce only one output value.

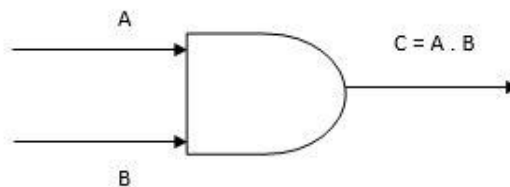
OR gate produces 1 output, when one of the inputs is 1. If inputs are 0, then the output will be also 0. It can be explained by taking an example of two switches connected in parallel.

The graphical symbol, algebraic expression and truth table of OR gate is as shown below:

Truth table:

Input		Output
A	B	$C = A + B$
0	0	0
0	1	1
1	0	1
1	1	1

Graphical Symbol:



Algebraic Expression is, $C = A + B$

Figure 4 OR gate truth table

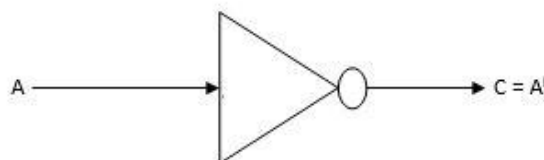
- NOT Gate:

The NOT Gate contains only one input value which produces only one output value. This gate is also known as an inverter. So, this circuit inverts the logical sense of a binary signal. It produces the complemented function. If the input is 1, then this gate will produce 0 as output and vice-versa. The graphical symbol, algebraic expression and truth table of a NOT gate is given below.

Truth table:

A	A'
0	1
1	0

Graphical Symbol:



Algebraic Expression is, $C = A'$

Figure 5 NOT gate truth table

- NAND Gate:

The NAND Gate contains two or more than two input values which produce only one output value. This gate is the combination of AND and NOT gates. This gate is a complement of AND function. This gate produces output 0, when all inputs are 1, otherwise, output will be 1.

The graphical symbol, algebraic expression and truth table of NAND gate is shown below:

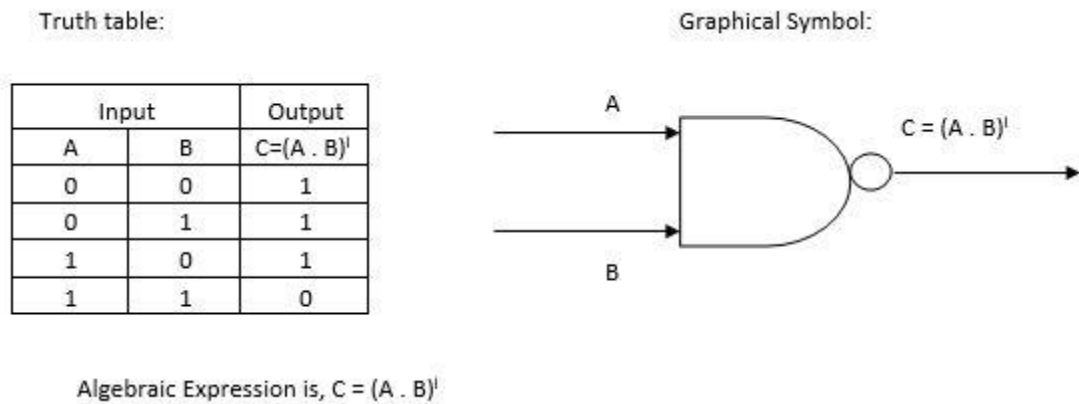


Figure 6 Nand gate truth table

- NOR Gate:

The NOR Gate contains two or more than two input values which produce only one output value.

This gate is a combination of OR and NOT gate.

The graphical symbol, algebraic expression and truth table of NOR gate are given below:

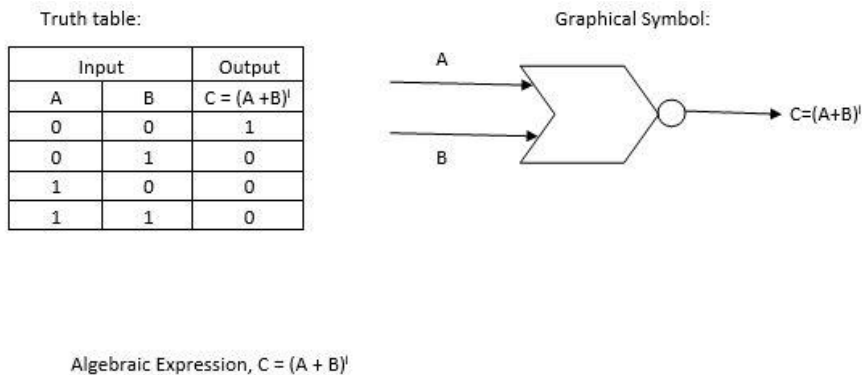


Figure 7 NOR gate Truth table

- Exclusive OR (X-OR) Gate:

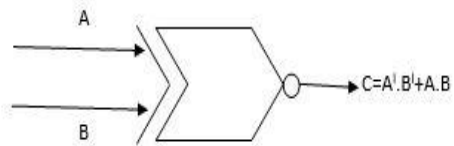
This gate contains two or more than two input values which produce only one output value. The graphical symbol of X-OR gate is similar to OR gate except for the additional curve line on the input side. This gate produces 1 as output, if any input is 1 and 0 if both inputs are either 1 or 0, otherwise its output is 0.

The graphical symbol, algebraic expression and truth table of X-OR gate is given below:

Truth table:

Input		Output
A	B	$C = A' \cdot B' + A \cdot B$
0	0	0
0	1	1
1	0	1
1	1	0

Graphical Symbol:



Algebraic Expression is, $C = A' \cdot B' + A \cdot B$

Figure 8 XOR gate Truth table

Algorithm

Step1: Start

Step2: Declare the variable a,b.

Step3: Read the variables.

Step4: Pass the variable a and b through the circuits.

- Pass the variable a and b through the first AND gate.
- Pass the variable a and b through the first XOR gate too.
- Pass the output of the first XOR gate to the second XOR gate and second AND gate.
- Pass the output of the first and second AND gate to the OR gate.

Step5: Display the sum of the logical gates of first circuit.

Step6: Pass the sum value or carry value to the next circuit through OR gate.

Step7: Pass the variable a and b through the second circuits.

- Pass the variable a and b to the first AND gate and XOR gate of second circuit.

- Pass the output of the first XOR gate and carry in value of OR gate of previous circuit through second XOR gate.
- Pass the carry in value of OR gate of previous circuit and output of the first XOR gate through second AND gate.
- Pass the output of both first and second AND gate through OR gate.

Step 8: Display the sum.

Step 9: Pass the carry value or sum value to next circuit through OR gate.

Step 10: Continue the same process.

Step 11: End.

Pseudo code:

A)Pseudocode of BinaryDecimal.py

Function binary(x):

lists **EQUALS** []

for i **in** range(8):

lists.**add**(x%2)

x **EQUALS** int(x/2)

return lists

Function reverse(x):

```

lists EQUALS []
y EQUALS -1
for i in range(len(x)):
    lists.add(x[y])
    y- EQUALS 1
return lists

```

```

Function decimal(x):
    y EQUALS 0
    for i in range(len(x)):
        y EQUALS y+(x[i]*2**i)
    return y

```

B)Pseudocode of FullAdder.py

Bring In logicgates

```

function FullAdder (x,y,z,Mylist3):
    y1 Equals logicgates.XOR(x,y)
    y2 Equals logicgates.AND(x,y)
    y3 Equals logicgates.AND(y1,z)
    y4 Equals logicgates.NOR(y2,y3)
    y5 Equals logicgates.OR(z,y1)
    y6 Equals logicgates.NAND(z,y1)
    Mylist3.append(logicgates.AND(y5,y6))
    z Equals logicgates.NOT(y4)

```

```
return z  
Display(z)
```

C)Pseudocode of Logicgates.py

function XOR (x,y):

```
    if x comparision with 1 and y comparision with 1:  
        return 0  
    elif x comparision with 0 and y comparision with 0:  
        return 0  
    else:  
        return 1
```

function AND (x,y):

```
    if x comparision with 0:  
        return 0  
    elif y comparision with 0:  
        return 0  
    else:  
        return 1
```

function NAND (x,y):

```
    if x comparision with 0:  
        return 1
```

elif y comparision with 0:

return 1

else:

return 0

function OR (x,y):

if x comparision with 1:

return 1

elif y comparision with 1:

return 1

else:

return 0

function NOR (x,y):

if x comparision with 1:

return 0

elif y comparision with 1:

return 0

else:

return 1

function NOT (x):

if x comparision with 1:

return 0

else:

return 1

D)Pseudocode of Main.py

Bring In binarydecimal

Bring In FullAdder

Function addition():

 check **Equals** "yes"

while check **EqualsEquals** "yes" or check**EqualsEquals**"YES":

 a **Equals** 257

while a<**Equals**0 | a>255:

while True:

 try:

 a **Equals** int(input("Enter the First Number : "))

if (a>**Equals**0 | a<256):

Display("The input is valid")

 end line

else:

Display("Value out of range")

except:

Display("Invalid Input")

b Equals 257

while b<**Equals**0 | b>255:

while True:

try:

 b **Equals** int(input("Enter the Second Number : "))

if (b>**Equals**0 | b<256):

Display("The input is valid")

end line

else:

Display("Value out of range")

except:

Display("Invalid Input")

g **Equals** a+b

if g>256:

Display("The sum is more than 255 therefore the program will be terminated.")

 exit()

else:

Display("The sum is" , g)

 check**Equals** "false"

MyList_list **Equals** binarydecimal.binary(a)

MyList2_list **Equals** binarydecimal.binary(b)

z **Equals** 0

MyList **Equals** []

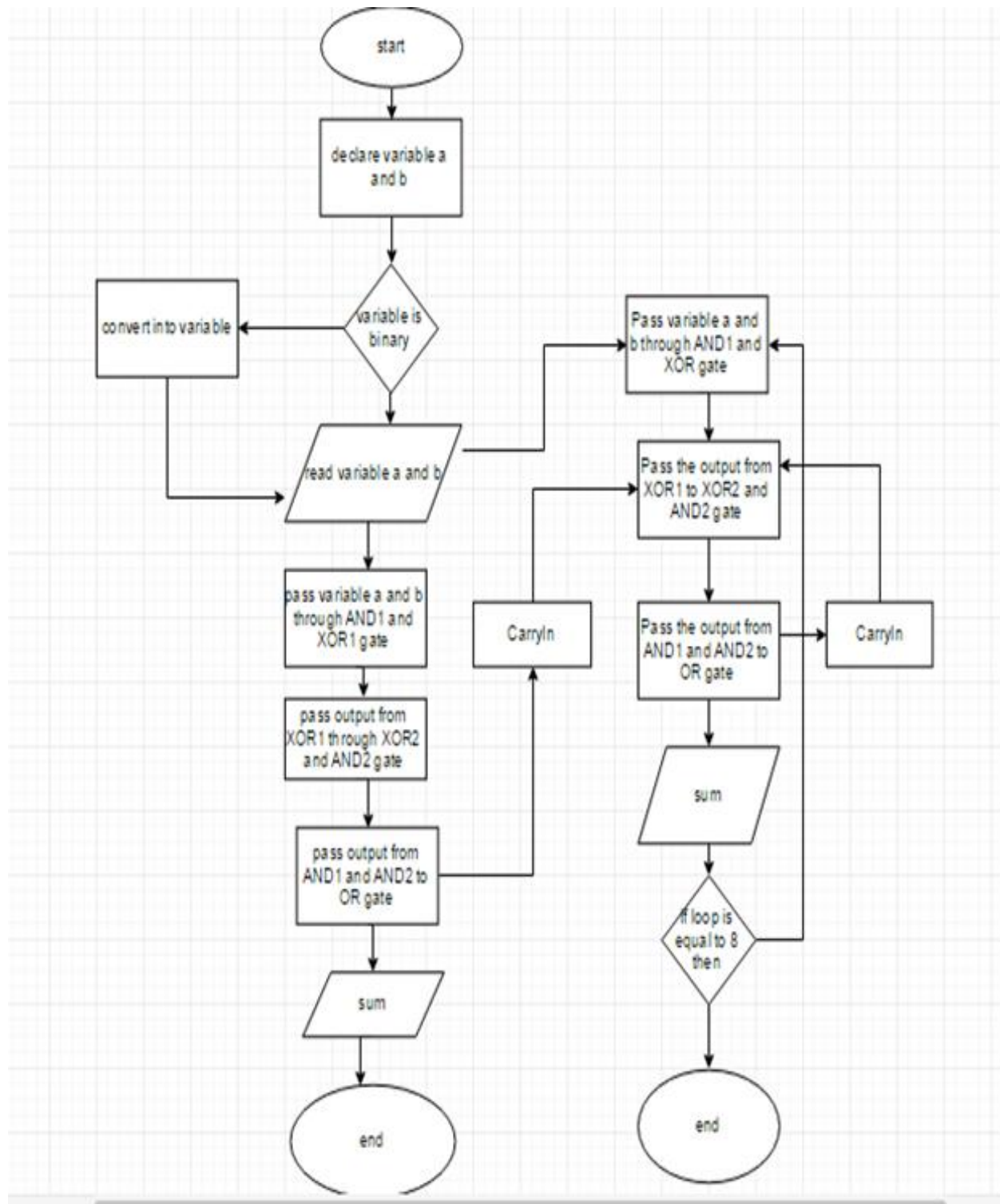
for i in **range** (len(MyList_list)):

```

    z Equals FullAdder.FullAdder(MyList_list[i], MyList2_list[i],z,MyList)
Display (a, binarydecimal.reverse(MyList_list))
Display (b, binarydecimal.reverse(MyList2_list))
Display ("-----")
Display (binarydecimal.decimal(MyList),binarydecimal.reverse(MyList))
check Equals input ("Do you want to Continue? (Yes/No)")
if check EqualsEquals "yes" or checkEqualsEquals"YES":
    addition()
else:
    exit()
addition()

```


Flowchart:



Above figure describes the Flowchart coding the half adder and full adder. In this flowchart, the different types of shape are used like Oval, Rectangle, Parallelogram, Diamond, Connector and Flow lines. Here, Oval shape starts and terminate the program, Rectangle shapes is used to performed the operation, Parallelogram shape is used to input and return the output, Diamond shape is used to give the decision to run the program, Connector is used to connect the different operation, Flow line is used to connect the different shapes.

Data Structures

A data structure is a specialized format for organizing and storing data. General data structure types include the array, the file, the record, the table, the tree, and so on. Any data structure is designed to organize data to suit a specific purpose so that it can be accessed and worked with in appropriate ways. In computer programming, a data structure may be selected or designed to store data for the purpose of working on it with various algorithms.

Well there are different kinds of data types I have also used some of them in order to achieve my goal and to make a complete a programme. Some of them data types that I have used are as follows:

Len (len)

Well it is used to calculate the words whenever the user inputs and also convert the value of binary.

String (str)

Firstly it is used for converting any integer value to strings so while making the programme I used this function whenever I need.

Integer (int)

It is used for converting the input data into integers.

Input (input)

It is used for getting user data so; I used these functions for getting any data or information from the users.

Bin (bin)

In the programming section it was used for converting the integer/decimal value into binary.

List (list)

As lists are compound data type which is made up of smaller parts and are flexible where we can add, change and remove. When there are a lot of values where we have to modify the values for that you have to work with data types. As an ordered sequence of elements, each item in a list can be called individually, through indexing.

In the programming section it was used for keeping the user inputs and the conversion values of binary into sequences

eval()

Eval() evaluates the passed string as a Python expression and returns the result. eval is almost always implemented with the same interpreter as normal code. This function can also be used to execute arbitrary code object. The return value is the result of the evaluated expression.

print()

print() function is not normally available as a built-in data structure. This function is used to print the statement which is very useful in every program.

Program

Main.py:

```
import binarydecimal
```

```
import FullAdder
```

```

def addition():

    check = "yes"

    while check == "yes" or check=="YES":

        a = int(input("Enter the First Number : "))

        if a>=0 | a<256:

            print("The input is valid")

        else:

            print("The input is invalid")

            exit()

        b = int(input("Enter the Second Number : "))

        if b>=0 | b<256:

            print("the input is valid")

        else:

            print("The input is invalid")

            exit()

        g = a+b

        if g>256:

            print("The sum is more than 255 therefore the program will be terminated.")

            exit()

        else:

            print("The sum is" , g)

            check= "false"

    MyList_list = binarydecimal.binary(a)

    MyList2_list = binarydecimal.binary(b)

    z = 0

```

```

MyList = []

for i in range (len(MyList_list)):

    z = FullAdder.FullAdder(MyList_list[i], MyList2_list[i],z,MyList)

print (a, binarydecimal.reverse(MyList_list))

print (b, binarydecimal.reverse(MyList2_list))

print ("-----")

print (binarydecimal.decimal(MyList),binarydecimal.reverse(MyList))

check = input ("Do you want to Continue? (Yes/No)")

if check == "yes" or check=="YES":

    addition()

else:

    exit()

addition()

```

```
Main.py - E:\Study Materials\CourseWorks\Information System\pYTHON FILES\Main.py (3.6.1)
File Edit Format Run Options Window Help

import binarydecimal
import FullAdder
def addition():
    check = "yes"
    while check == "yes" or check=="YES":

        a = 257
        while a<=0 | a>255:
            try:
                a = int(input("Enter the First Number : "))
                if (a>=0 | a<256):
                    print("The input is valid")
                    continue
                else:
                    print("Value out of range")
            except:
                print("Invalid Input")

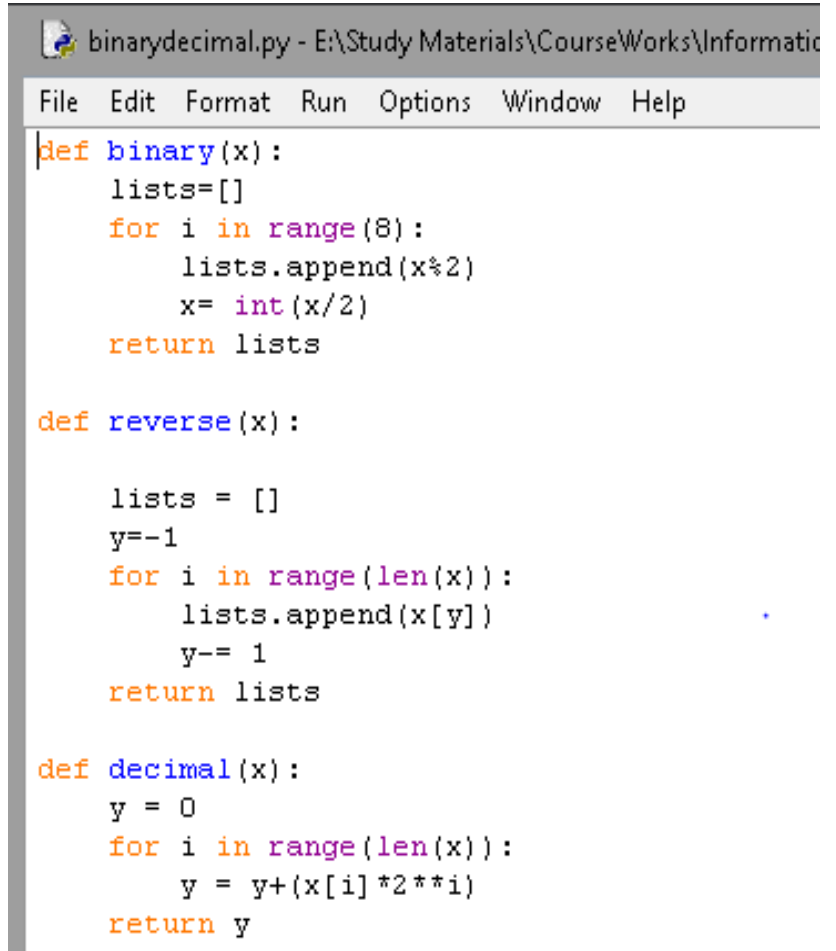
        b = 257
        while b<=0 | b>255:
            try:
                b = int(input("Enter the Second Number : "))
                if (b>=0 | b<256):
                    print("The input is valid")
                else:
                    print("Value out of range")
            except:
                print("Invalid Input")
        g = a+b
        if g>256:
            print("The sum is more than 255 therefore the program will be terminated.")
            exit()
        else:
            print("The sum is" , g)
            check= "false"
        MyList_list = binarydecimal.binary(a)
        MyList2_list = binarydecimal.binary(b)
        z = 0
        MyList = []
        for i in range (len(MyList_list)):
            z = FullAdder.FullAdder(MyList_list[i], MyList2_list[i],z,MyList)
        print (a, binarydecimal.reverse(MyList list))
```

Figure 9 Main.py Code

BinaryDecimal.py:

```
def binary(x):  
    lists=[]  
    for i in range(8):  
        lists.append(x%2)  
        x= int(x/2)  
    return lists  
  
def reverse(x):  
  
    lists = []  
    y=-1  
    for i in range(len(x)):  
        lists.append(x[y])  
        y-= 1  
    return lists
```

```
def decimal(x):  
    y = 0  
    for i in range(len(x)):  
        y = y+(x[i]*2**i)  
    return y
```



```
binarydecimal.py - E:\Study Materials\CourseWorks\Informati  
File Edit Format Run Options Window Help  
def binary(x):  
    lists=[]  
    for i in range(8):  
        lists.append(x%2)  
        x= int(x/2)  
    return lists  
  
def reverse(x):  
  
    lists = []  
    y=-1  
    for i in range(len(x)):  
        lists.append(x[y])  
        y-= 1  
    return lists  
  
def decimal(x):  
    y = 0  
    for i in range(len(x)):  
        y = y+(x[i]*2**i)  
    return y
```

Figure 10 Binary Decimal.py

FullAdder.py:

```
def binary(x):
```

```
    lists=[]
```

```
    for i in range(8):
```

```
        lists.append(x%2)
```

```
        x= int(x/2)
```

```
    return lists
```

```
def reverse(x):
```

```
    lists = []
```

```
    y=-1
```

```
    for i in range(len(x)):
```

```
        lists.append(x[y])
```

```
        y-= 1
```

```
    return lists
```

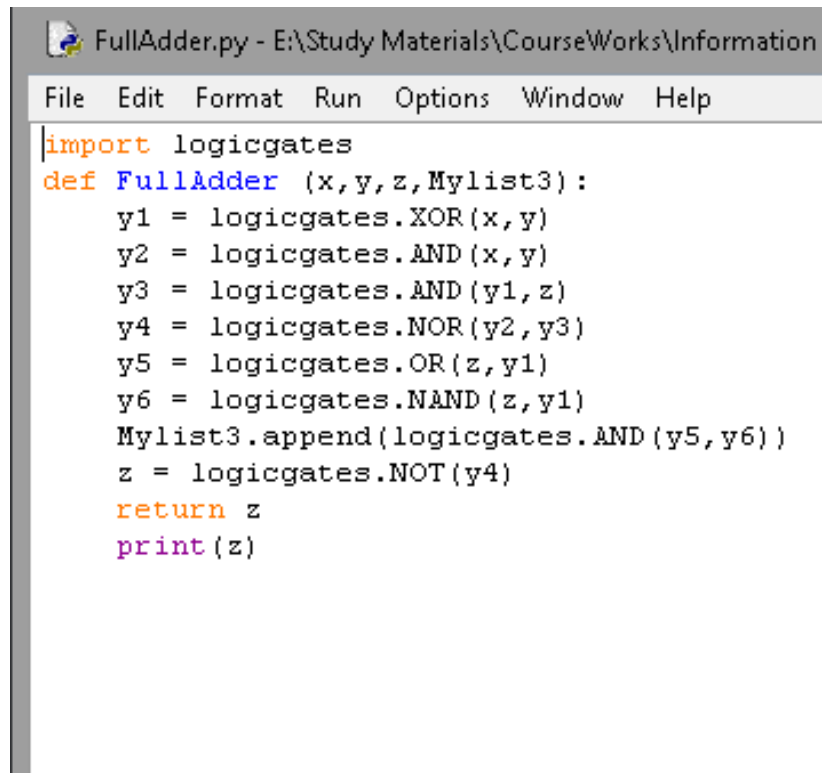
```
def decimal(x):
```

```
    y = 0
```

```
    for i in range(len(x)):
```

```
        y = y+(x[i]*2**i)
```

```
    return y
```



```
FullAdder.py - E:\Study Materials\CourseWorks\Information
File Edit Format Run Options Window Help
import logicgates
def FullAdder (x,y,z,Mylist3):
    y1 = logicgates.XOR(x,y)
    y2 = logicgates.AND(x,y)
    y3 = logicgates.AND(y1,z)
    y4 = logicgates.NOR(y2,y3)
    y5 = logicgates.OR(z,y1)
    y6 = logicgates.NAND(z,y1)
    Mylist3.append(logicgates.AND(y5,y6))
    z = logicgates.NOT(y4)
    return z
    print(z)
```

Figure 11 FullAdder.py

LogicGates.py:

```
def XOR (x,y):
```

```
    if x == 1 and y==1:
```

```
        return 0
```

```
    elif x==0 and y==0:
```

```
        return 0
```

```
    else:
```

```
        return 1
```

```
def AND (x,y):
```

```
    if x==0:
```

```
        return 0
```

```
    elif y==0:
```

```
        return 0
```

```
    else:
```

```
        return 1
```

```
def NAND (x,y):
```

```
    if x==0:
```

```
        return 1
```

```
    elif y==0:
```

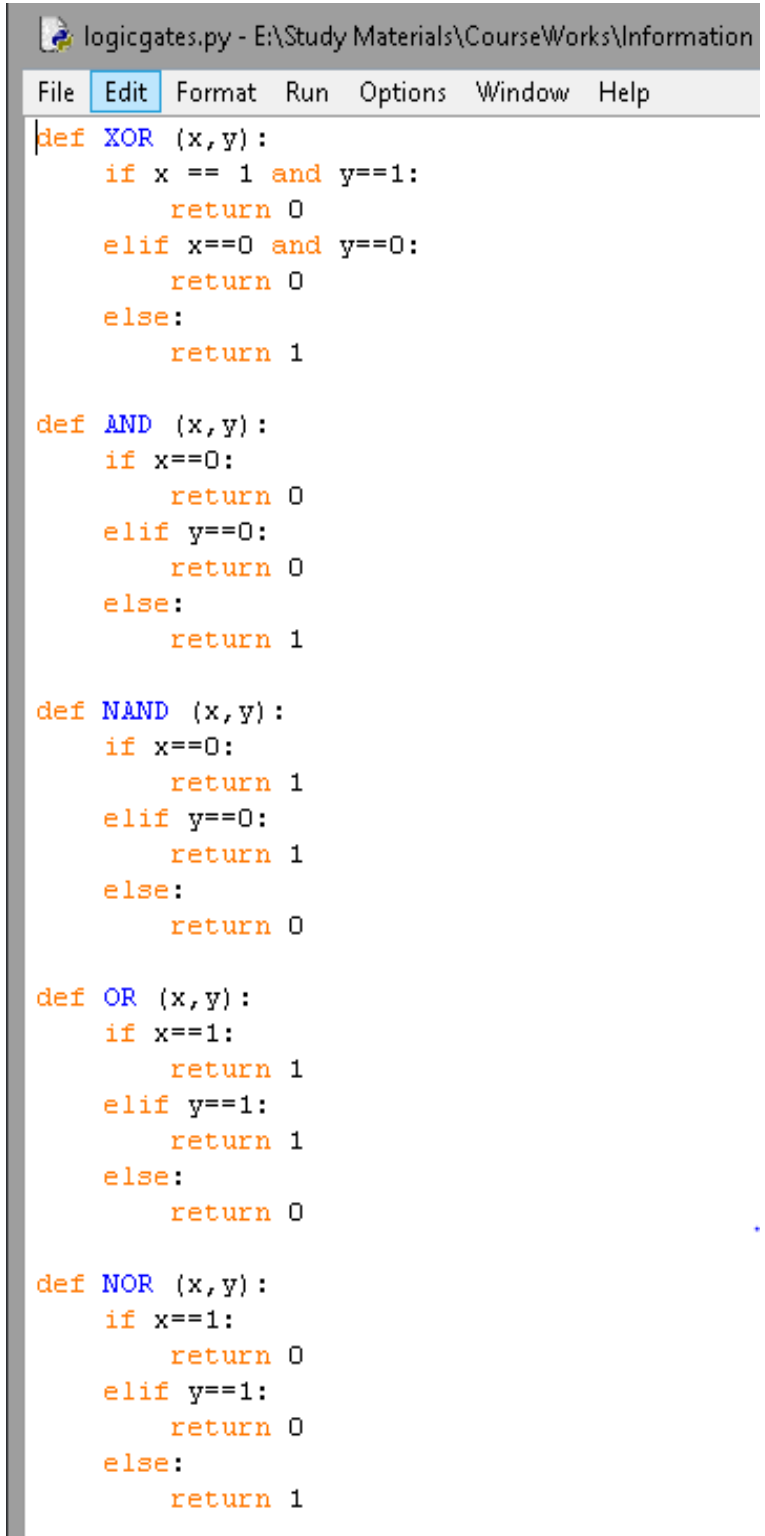
```
        return 1
```

```
    else:
```

```
        return 0
```

```
def OR (x,y):
```

```
    if x==1:
```



```
logicgates.py - E:\Study Materials\CourseWorks\Information
File Edit Format Run Options Window Help
def XOR (x,y):
    if x == 1 and y==1:
        return 0
    elif x==0 and y==0:
        return 0
    else:
        return 1

def AND (x,y):
    if x==0:
        return 0
    elif y==0:
        return 0
    else:
        return 1

def NAND (x,y):
    if x==0:
        return 1
    elif y==0:
        return 1
    else:
        return 0

def OR (x,y):
    if x==1:
        return 1
    elif y==1:
        return 1
    else:
        return 0

def NOR (x,y):
    if x==1:
        return 0
    elif y==1:
        return 0
    else:
        return 1
```

Figure 12 LogicGates.py

```
        return 1
    elif y==1:
        return 1
    else:
        return 0

def NOR (x,y):
    if x==1:
        return 0
    elif y==1:
        return 0
    else:
        return 1

def NOT (x):
    if x==1:
        return 0
    else:
        return 1
```

Research

Books

- Learn python the hard way

Originally published: September 19, 2013

Author: Zed Shaw

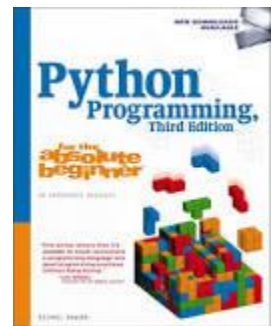
This book gave me the basic understanding about the computer number system by which it helped a lot while creating the logic gate and truth table.

- Python programming for the absolute beginner

Originally published: 2003

Author: J. Michael Dawson

This is the book is Developed by computer science instructors; books in the "for the absolute beginner" series teach the principles of programming through simple game creation. You will acquire the skills that you need for practical Python programming applications and will learn how these skills can be put to use in real-world scenarios.

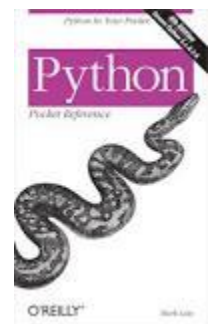


- Python pocket reference

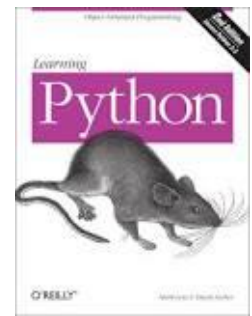
Originally published: 1998

Author: Mark Lutz

Python is optimized for quality, productivity, portability, and integration. Hundreds of thousands of Python developers around the world rely on Python for general-purpose tasks, Internet scripting, systems programming, user interfaces, and product customization. Available on all major computing platforms, including commercial versions of Unix, Linux, Windows, and Mac OS X, Python is portable, powerful and remarkable easy to use.



- Learning python
Originally published: 1999
Authors: Mark Lutz, David Ascher
Original language: English



This book has covered all the essential needs for the programmer. We can find anything here details.

Journals

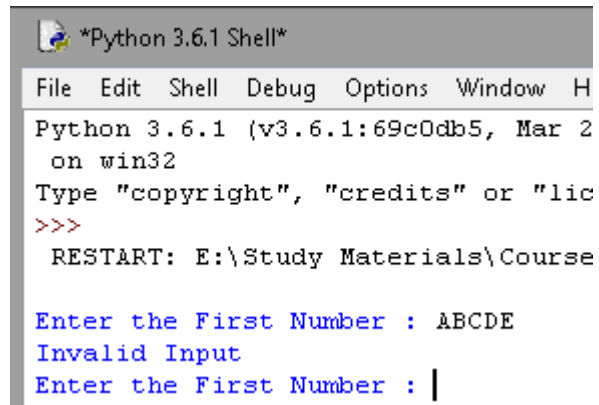
- Linux journals
- The python papers
- Early journals
- Python humor

Websites

- <https://www.python.org>
- <https://www.wolfram.com/system-modeler/examples/more/electrical-engineering/8-bit-adder>
- <https://psyborgs.github.io/tutorials/digging-into-jstors-early-journals/>
- <https://www.python.org/doc/humor/>
- <https://www.topcoder.com/community/data-science/data-science-tutorials/the-importance-of-algorithms/>

Testing

Alphabet User input



```
*Python 3.6.1 Shell*
File Edit Shell Debug Options Window H
Python 3.6.1 (v3.6.1:69c0db5, Mar 2
on win32
Type "copyright", "credits" or "lic
>>>
RESTART: E:\Study Materials\Course

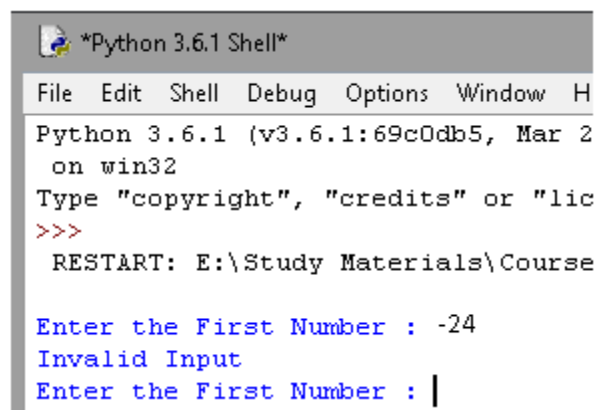
Enter the First Number : ABCDE
Invalid Input
Enter the First Number : |
```

Figure 13 Alphabet Test

1 st User Input Value	2 nd User Input Value	Result/Output
ABCDE	-	Invalid literal for int () with base 10.

Table 1 Alphabet Test

Passing Negative Value



```
*Python 3.6.1 Shell*
File Edit Shell Debug Options Window H
Python 3.6.1 (v3.6.1:69c0db5, Mar 2
on win32
Type "copyright", "credits" or "lic
>>>
RESTART: E:\Study Materials\Course

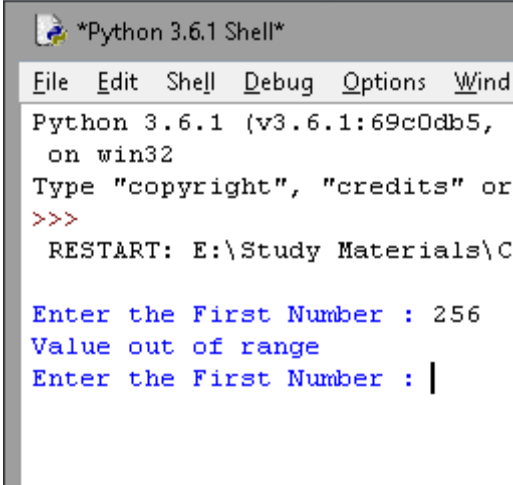
Enter the First Number : -24
Invalid Input
Enter the First Number : |
```

Figure 14 Negative Value Test

1 st User Input Value	2 nd User Input Value	Result/Output
-24	22	Invalid Input

Table 2 Negative Value Test

Out Of Range Value From User



```
*Python 3.6.1 Shell*
File Edit Shell Debug Options Wind
Python 3.6.1 (v3.6.1:69c0db5,
on win32
Type "copyright", "credits" or
>>>
RESTART: E:\Study Materials\C

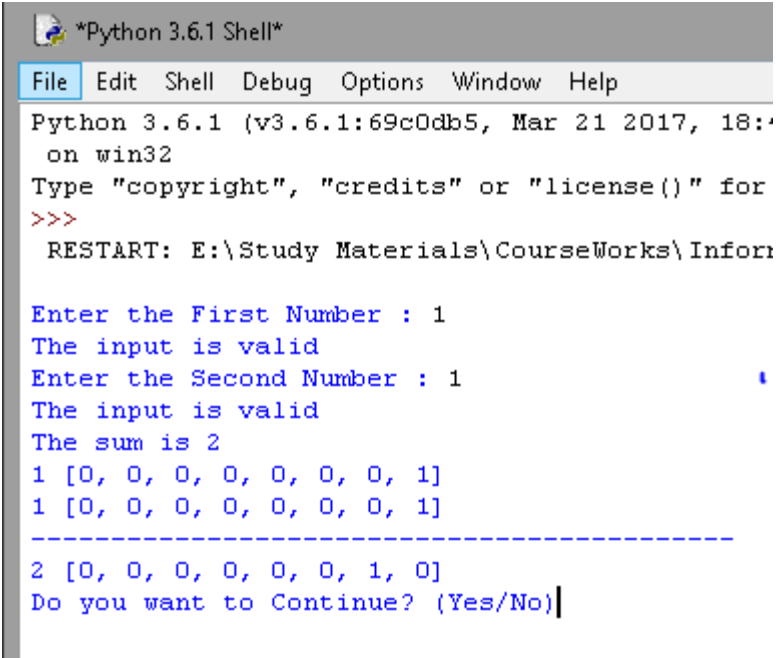
Enter the First Number : 256
Value out of range
Enter the First Number : |
```

Figure 15 Out of range Value Test

1 st User Input Value	2 nd User Input Value	Result/Output
256	256	Value out of Range:

Table 3 Out of range Value Test

User Input of 1 and 1



```
*Python 3.6.1 Shell*
File Edit Shell Debug Options Window Help
Python 3.6.1 (v3.6.1:69c0db5, Mar 21 2017, 18:
on win32
Type "copyright", "credits" or "license()" for
>>>
RESTART: E:\Study Materials\CourseWorks\Infor

Enter the First Number : 1
The input is valid
Enter the Second Number : 1
The input is valid
The sum is 2
1 [0, 0, 0, 0, 0, 0, 0, 0, 1]
1 [0, 0, 0, 0, 0, 0, 0, 0, 1]
-----
2 [0, 0, 0, 0, 0, 0, 0, 1, 0]
Do you want to Continue? (Yes/No)|
```

Figure 16 input 1 and 1

1 st User Input Value	2 nd User Input Value	Result/Output
1	1	00000010

Table 4 1 on 1 input

Value crosses from setted limited value and passed with negative integer

```

Python 3.6.1 Shell*
File Edit Shell Debug Options Window Help
Python 3.6.1 (v3.6.1:69c0db5, Mar 21 2017)
on win32
Type "copyright", "credits" or "license()" for more
>>>
RESTART: E:\Study Materials\CourseWork\Python\3.6.1\Python 3.6.1 Shell*

Enter the First Number : 12
The input is valid
Enter the Second Number : -500
Invalid Input
Enter the Second Number : |

```

Figure 17 Value crosses from setted limited value and passed with negative integer

1 st User Input Value	2 nd User Input Value	Result/Output
12	-500	Invalid Input

Table 5 Value crosses passed with negative integer

Both value is 0

```

Enter the First Number : 0
The input is valid
Enter the Second Number : 0
The input is valid
The sum is 0
0 [0, 0, 0, 0, 0, 0, 0, 0, 0]
0 [0, 0, 0, 0, 0, 0, 0, 0, 0]
-----
0 [0, 0, 0, 0, 0, 0, 0, 0, 0]
Do you want to Continue? (Yes/No)|

```

Figure 18 both value 0 test

1 st User Input Value	2 nd User Input Value	Result/Output
0	0	00000000

Table 6 Both value is 0

Conclusion

Well, with all that afford finally I am glad to say that I had completed the course work. In my opinion I think the module information system is very essential for all students across the world. This module is important because it mainly focuses in the programming language and we all know that in this 21st century if we talk about the technology it is progressing rapidly. Human works are being reduced and works are becoming easier and faster that's why a person having knowledge of programming language (PYTHON) is more demanding in today's world.

The course work given by which was to create the circuit this task made me to understand the basic rules of gates and how they are in-related with the program or how they link up with the programs. This task made more familiar with the algorithm I came to know that algorithm is the step by step process of writing before starting any program. Basically algorithm reviews the python programming language and also plays the role in solving the problems. While doing this course work I was able to learn about the guidelines and important codes about the structure of programming. I had to write the pseudocode as well as I had learn about pseudocode in java already so I felt easy to write the code. It is the process of just writing the codes into the readable format so that everyone can understand what the programme means.

To be honest it took about 1 week for me to complete the coding. The coding part was most difficult part for me. I was unable to make the programme that our module teacher had said us to make I was disappointed. While I was writing the codes I have many problems like adding two integers and converting them into the binary and getting the result in 8-bit format also implementing gates but I was not getting the result. With the help of research and taking the advice from module teachers finally I was able to overcome that problem and was successful to make the programme.

Bibliography

Book named “Systems Analysis and Design Methods “by Garry B. Shelly, Thomas J. Cashman and Harry J. Rosenblatt

Book named “Computer System Architecture” by M. Morris Mano

Book named “Python Programming” by Michael Davon

Journal from A.Fayed and M.Bayou MI, “A low power 10-tranistor adder cell for embedded architectures, “/EEE Symposium of Circuits and Systems, Sydney, Australia, pp.226-229, May 2001

Journal from N.Zhuang and H.Wu,” A new design of the CMOS full adder”, IEEE J. Solid state circuits, Vol.27,pp.840-844, May 1992

Journal from A.M. Shams and MagdyA.Bayoumi, “A Novel High Performances CMOS 1-Bit Full Adder Cell,”IEEE Trans. Circuits and Systems-II, Vol.47No.5, May 2000

Journal from A.M. Shams and MagdyA.Bayoumi, “A Novel High Performances CMOS 1-Bit Full Adder Cell,”IEEE Trans. Circuits and Systems-II, Vol.47No.5, May 2000